A. Algorithm Overview

Algorithm 1: HyperSegNAS Training

Networks: image encoder: $\mathcal{M}$, Meta-Assistant Network: $\mathcal{H}$, super-net: $\mathcal{F}$
Input: sampled input image: $I_i \in \mathbb{R}^{96\times96\times96}$, sampled architecture: $a_i \in \mathbb{N}^{L\times E\times O}$, ground truth: $S_{gt} \in \mathbb{N}^{96\times96\times96}$, total layers: $L = 12$, total edges between columns: $E = 10$, total operations: $O = 2$, super-net training iter.: $N_1 = 160000$, annealing training iter.: $N_2 = 20000$
Output: segmentation mask: $S \in \mathbb{N}^{96\times96\times96}$
for $i = 1$ **to** N_1 **do**
  $l_{\text{image}} = \mathcal{M}(I_i)$
  $l_{\text{arch}} = \text{flatten}(a_i)$
  $\omega_{\mathcal{H}_\theta}^{a_i,I_i} = \mathcal{H}(l_{\text{arch}} \oplus l_{\text{image}})$
  $\omega_{\text{active}} = \omega_{\mathcal{H}_\theta}^{a_i,I_i}(l,e,c) * \omega_a(l,e,c)$
  $S = \mathcal{F}(I_i; a_i, \omega_{\text{active}})$
  $\mathcal{L}_{\text{train}}(S, S_{gt}) = \mathcal{L}_{\text{dice}}(S, S_{gt}) + \mathcal{L}_{\text{ce}}(S, S_{gt})$
  $\omega_{\text{active}} := \omega_{\text{active}} - \alpha \frac{\delta \mathcal{L}_{\text{train}}}{\delta \omega_{\text{active}}}$
end
for $i = 1$ **to** N_2 **do**
  $l_{\text{image}} = \mathcal{M}(I_i)$
  $l_{\text{arch}} = \text{flatten}(a_i)$
  $\omega_{\mathcal{H}_\theta}^{a_i,I_i} = \mathcal{H}(l_{\text{arch}} \oplus l_{\text{image}})$
  $\lambda = \frac{i}{N_2}$
  $\omega_{\text{active}} = (\lambda\omega_{\text{fixed}} + (1-\lambda)\omega_{\mathcal{H}_\theta}^{a_i,I_i}) * \omega_a)$
  $S = \mathcal{F}(I_i; a_i, \omega_{\text{active}})$
  $\mathcal{L}_{\text{train}}(S, S_{gt}) = \mathcal{L}_{\text{dice}}(S, S_{gt}) + \mathcal{L}_{\text{ce}}(S, S_{gt})$
  $\omega_{\text{active}} := \omega_{\text{active}} - \alpha \frac{\delta \mathcal{L}_{\text{train}}}{\delta \omega_{\text{active}}}$
end

We provide a detailed description of the training pipeline for HyperSegNAS in Algorithm 1, which includes training the super-net with $\mathcal{H}$ in N_1 iterations and annealing away $\mathcal{H}$ in N_2 iterations. For clarity, we denote the super-net as $\mathcal{F}(*; a, \omega_a)$, where the forward function involves the inferenced architecture and the corresponding weights. The flatten$(*)$ function converts the one-hot architecture matrix $a \in \mathbb{N}^{L\times E\times O}$ to the vector $l_{\text{arch}} \in \mathbb{N}^{L\cdot E}$, where each element in l_{arch} indicates either inactive, a skip connection, or a $3\times3\times3$ convolution. Training loss is based on an equal combination of dice and cross-entropy loss. The super-net training schedule is similar to DiNTS [16], where we use 1000 iterations for warm-up, during which the learning rate α linearly increases from 0.025 to 0.2. After warm-up, learning rate decreases by half at 20%, 40%, 60%, and 80% of N_1. During annealing, the learning rate is fixed at 0.0016. Similarly, during quick fine-tuning over individual architectures, learning rate is fixed at 0.0016. The architectures are fine-tuned over 5000 iterations.

B. Network Architecture

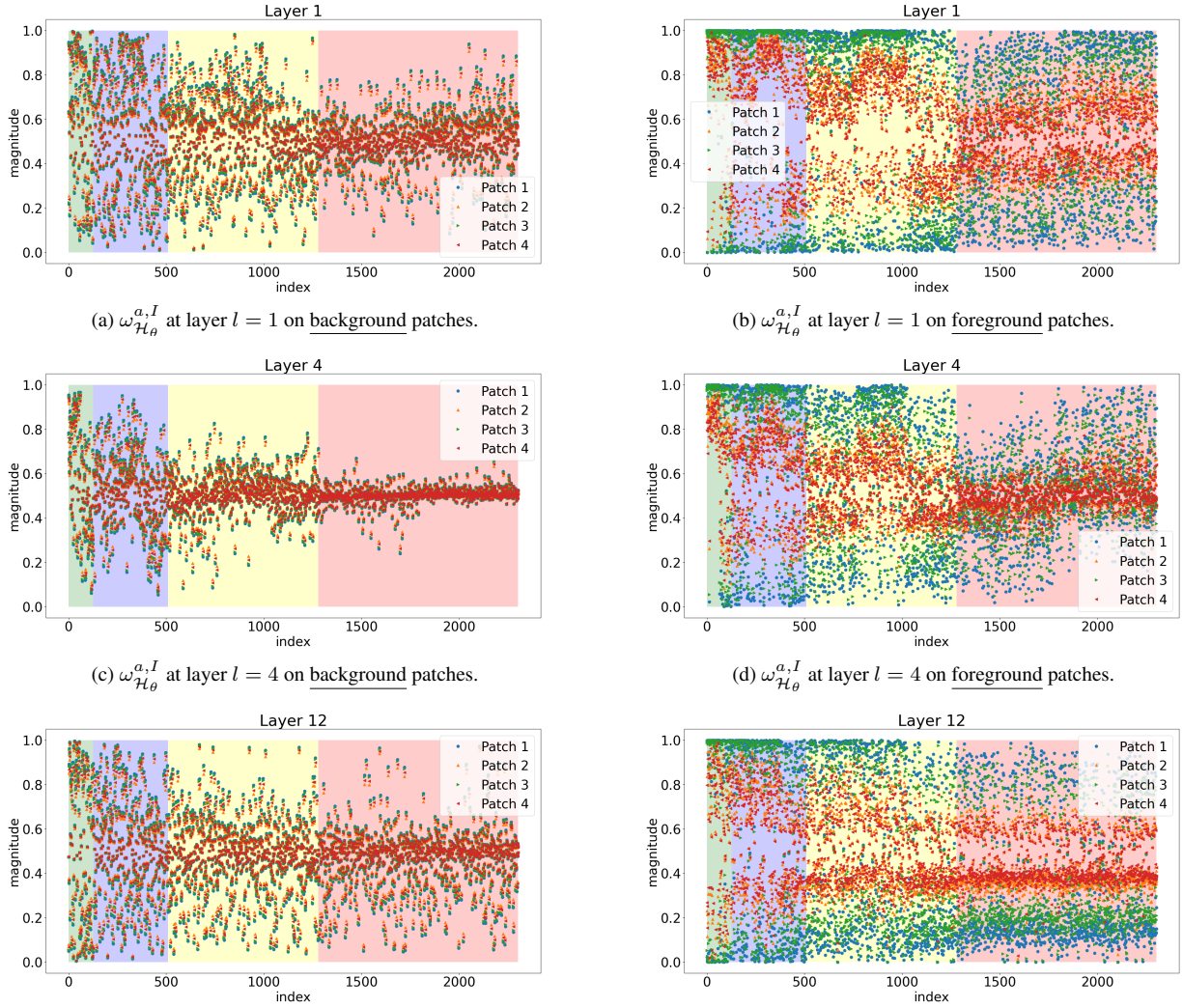
We provide the network architectures of the image encoder $\mathcal{M}$ and the Meta Assistant Network (MAN) in Table 4 and Table 5. In the network tables, N_c denotes the number of output channels, C' denotes the channel dimension of generated features F^{LR}. We use 'K#-C#-S#-P#' to denote the configuration of the convolution layers, where 'K', 'C', 'S' and 'P' stand for the kernel, input channel, stride and padding size, respectively.

Name	N_c	Description
INPUT	1	Input I
CONV0	16	K3-C1-S2-P1
IN		InstanceNorm3d
RELU		
CONV1	32	K3-C16-S2-P1
IN		InstanceNorm3d
RELU		
CONV2	64	K3-C32-S2-P1
IN		InstanceNorm3d
RELU		
CONV3	128	K3-C64-S2-P1
IN		InstanceNorm3d
RELU		
CONV4	256	K3-C128-S2-P1
IN		InstanceNorm3d
RELU		
CONV5	256	K3-C256-S1-P0
RELU		

Table 4. Network architecture of $\mathcal{M}$.

Name	N_c	Description
INPUT	378	Input $l_{\text{arch}} \oplus l_{\text{image}}$
FC0	2048	C378
RELU		
FC1	4096	C2048
RELU		
FC2	8192	C4096
RELU		
FC3	16384	C8192
RELU		
FC4	27648	C16384
SIGMOID		

Table 5. Network architecture of $\mathcal{H}$.



(a) $\omega_{\mathcal{H}_\theta}^{a,I}$ at layer $l = 1$ on background patches.

(b) $\omega_{\mathcal{H}_\theta}^{a,I}$ at layer $l = 1$ on foreground patches.

(c) $\omega_{\mathcal{H}_\theta}^{a,I}$ at layer $l = 4$ on background patches.

(d) $\omega_{\mathcal{H}_\theta}^{a,I}$ at layer $l = 4$ on foreground patches.

(e) $\omega_{\mathcal{H}_\theta}^{a,I}$ at layer $l = 12$ on background patches.

(f) $\omega_{\mathcal{H}_\theta}^{a,I}$ at layer $l = 12$ on foreground patches.

Figure 7. Visualization of $\omega_{\mathcal{H}_\theta}^{a_i,I_i}$ generated from $\mathcal{H}$. The weights within the green regions correspond to the highest resolution features, $F_{\downarrow 2}^l$, the blue region corresponds to weights for $F_{\downarrow 4}^l$, the yellow region corresponds to weights for $F_{\downarrow 8}^l$, and the red region corresponds to weights for $F_{\downarrow 16}^l$. Clear differences can be observed between weights generated from foreground and background patches.

C. HyperNet Weights Visualization

We provide additional visualizations on $\omega_{\mathcal{H}_\theta}^{a_i,I_i}$ generated from $\mathcal{H}$. There are $L = 12$ layers in our search space, and each layer contains $E = 10$ possible edges each with different channel numbers depend on the feature resolution. We select eight image patches from a single volume, four of which are background and the rest four are foreground. We record the corresponding $\omega_{\mathcal{H}_\theta}^{a_i,I_i}$ and plot the weight values in layer $\{1, 4, 12\}$ in Fig. 7. The weights are plotted following the order of the feature resolutions. We can make several observations: (1) weights in different layers have different distributions; (2) weights generated from background image patches are similar to each other within the same layer; (3) weights generated from foreground image patches are different to each other and to the background patches, within the same layer. This observation indicates that $\mathcal{H}$ learns an efficient channel-weighting mechanism that can differentiate foreground and background. We also observe that the weights for low-resolution features tend more towards 0.5, which may imply that these features are less important to learning due to their scale. We also observe that the weights of low-resolution feature tends to be 0.5, which may imply that these features are less important for learning due to their scale. Finally, it is possible to increase the sensitivity to foreground labels by modifying $\omega_{\mathcal{H}_\theta}^{a_i,I_i}$, which may be interesting for future investigations.